Summer 6-12-2015

Cooperative 3-D Map Generation Using Multiple UAVs

Andrew Erik Lawson
University of Connecticut - Storrs, andrew.lawson@uconn.edu

COOPERATIVE 3-D MAP GENERATION USING MULTIPLE UAVS

Andrew Lawson
University of Connecticut
Computer Science and Engineering

June 12, 2015

Abstract

This report aims to demonstrate the feasibility of building a global 3-D map from multiple UAV robots in a GPS-denied, indoor environment. Presented are the design of each robot and the reasoning behind choosing its hardware and software components, the process in which a single robot obtains a individual 3-D map entirely onboard, and lastly how the mapping concept is extended to multiple robotic agents to form a global 3-D map using a centralized server. In the latter section, this report focuses on two algorithms, *Online Mapping* and *Map Fusion*, developed to facilitate the cooperative approach. A limited selection of experiments and test results are also presented to demonstrate application of these algorithms in a real-world setting.

Acknowledgements

I'd like to thank Prof. Shalabh Gupta, Prof. Ashwin Dani, and Prof. Robert McCartney for their advice and support in writing this report. I also really want to thank Kris Balisciano and Chung Yang for all their hard work on the senior design project this year. Finally, I want to thank Kehan Zhou for all his help in making this possible.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

1.1 OVERVIEW

In the modern field of robotics, the ability for a robot to move about the environment, regardless of the medium, has proven to be fairly trivial. However, the ability for a robot to track its own position and make deterministic decisions about it's own location in a foreign environment remains a fundamental, but critical problem. In most typical robotic experiments, the use of external measurement systems (such as GPS or motion capture cameras) allows researchers to capture the exact position of a robot in its environment and focus on other areas of interest. In the past couple decades, successful algorithms called SLAM[1] have allowed singular robots to determine their own position in a world frame and construct a 3-D map from collected positional data. However, despite these advances, the ability for multiple UAV[2] robots to cooperatively determine their environmental boundaries is an area that remains relatively under-developed. By taking this cooperative approach with multiple robots, teams of coordinated robots could efficiently explore unaccessible areas, complete team-based tasks (like lifting large objects), conduct dangerous search and rescue missions, etc. This report explores the feasibility of mapping an indoor environment using multiple UAV robots where external positioning systems are not available. It should be noted that the navigation component of a robot using a SLAM system was not the focus of this project - the experiments and algorithms within instead concentrate on the computational and communicative aspects of developing a global map in such an environment.

1.2 LITERATURE REVIEW

In recent research projects with quadcopters, a wide variety of methods have been used to localize a robot in their environment, including (but not limited to) the use of high-resolution laser scanners [2, 3], monocular

[1] Simultaneous-Localization-And-Mapping: algorithm that uses sensor data to independently determine a position relational to an environment and simultaneously create a virtual map
[2] UAV: unmanned aerial vehicle

cameras [4], and "RGB-D"[3] cameras as a combination of the two [5]. A laser has been used to generate a point cloud and pull out contours that trace the environment [2], a monocular camera to exploit texture gradients and continuations that can translated into depth estimates [4], and Microsoft Kinects (RGB-D cameras) to associate laser depth with image data [5]. Lasers capable of such precision and frequency are economically infeasible, and additionally, a pure laser approach can result in large margins of error. Using a combination of the two, by means of an RGB-D camera, is responsive, inexpensive, and produces results of reasonable reliability.

In recent years, researchers have attempted cooperative mapping methods similar to what's discussed in this report. Walter et al. showed experiments using cooperative SLAM in underwater vehicles [6]. Riazuelo et al. developed the C^2TAM cloud framework for cooperative tracking and mapping using UAVs [7], where mapping and optimization is computed in the cloud. A few researchers have even demonstrated decentralized SLAM methods, but either do not provide total three-dimensional capability or use external positioning sensors. For instance, Leung et al. used a VICON camera system [8] to provide true location data to decentralized robotic agents. Cunningham et al. implemented truly autonomous decentralized SLAM, but only focused on 2-D, planar data analysis [9].

This project attempts to build on this precedent by demonstrating cooperative mapping with UAVs as opposed to ground or aquatic robots. In addition, the mapping process is computed completely onboard the robot (as opposed to the cloud) and a centralized server is only used with the introduction of the cooperative component. Lastly, the robots used in this report are capable of building and communicating a completely three dimensional map as opposed to 2.5D or completely two dimensional data.

[3] RGB-D: red-blue-green-depth

2

ROBOT DESIGN

2.1 DESIGN DECISIONS

Several key design decisions arose when choosing a direction for robot design and selection of components. The most significant choice for the robot was choosing an architecture for the onboard computational unit - specifically an ARM or x86 computer. The second key design dilemma (as mentioned briefly in the literature review) involved the selection of a laser-scanning sensor or an RGB-D camera. As shown below, both these choices had great effect upon the path of experimentation and results obtained.

2.1.1 *ARM vs. x86*

The main advantage of choosing an x86 board (found in most desktop or laptop computers) is compatibility. Almost all software packages are available in prebuilt, binary form on package managers (such as Ubuntu's apt-get) and setting up the computing environment can be done with a few simple installation commands. This convenience, unfortunately, comes at a financial and dimensional price - most small form-factor x86 boards are extremely expensive and cheaper options are considerably larger in size. These tradeoffs make selection of such a board infeasible for an aerial drone where space is extremely limited and minimization of cost is a primary concern. As a result, ARM boards (like those in smartphones) are much more attractive - they offer extremely small form-factors, yet remain very affordable. The issue that arises, of course, is that of compatibility - many programs or libraries are not (immediately) available on an ARM installation and require manual compilation of source code using an ARM-based compiler and possible changes to the source code itself first. For this project, an ARM board was selected for the reasons listed above.

2.1.2 *Laser vs. RGB-D Camera*

The two major choices of sensors for robotic vision are using a laser scanner or an RGB-D camera. Much like the x86 computers, lasers are a terrific choice when cost is not a concern - they have wide field of view

(some providing between 270 and 360 degrees), can scan very quickly (40 scans per second), and have exceptional range (around 30 meters). However, a single unit can cost upwards of five thousand dollars and using only depth data can result in a large accumulation of error. RGB-D cameras, like the Microsoft Kinect, are very cheap sensors in comparison (the Kinect costing only $99 per unit). Unlike the laser, these sensors provide both depth and camera image data - less error is produced as a result of the ability to track image features across frames. However, they are slower (maximum of 30 scans per second) and require significantly more bandwidth for data transmission. Due to cost and error concerns, an RGB-D camera was selected for the experiments in this report.

2.2 HARDWARE

2.2.1 *Components*

The hardware design of the robot was focused on using low-cost, off-the-shelf components (see Table 1). The approximate cost of a built robot was $1045, significantly cheaper than pre-built alternatives such as the AscTec Pelican used at MIT [2] and University of Washington [10] with similar project requirements.

Table 1: Table of Components

Component	Description	Cost
DJI F450 Frame	Chassis	$190
DJI 1045 Propellers	Props	$6
DJI 2212 Motors	Actuators	$92
DJI 30A ESCs	Speed Controllers	$80
12V and 5V Regulators	Power Reg.	$35
Zippy 2450 mAh LiPo	Battery	$22
64GB ODROID XU3	ARM SBU	$266
Pixhawk PX4	Controller	$227
Microsoft Kinect	Visual Sensor	$99
MaxSonar EZ0	Sonar Sensor	$28

The robot used a stock F450 frame from DJI, which was not modified in any way - all components were mounted on existing areas using screws or adhesive. The onboard computational unit chosen was the ODROID XU3 by HardKernel, which has a Samsung Exynos5422 2.0Ghz quad-core CPU, 2GB DDR3 RAM, 64GB eMMC flash storage, USB 3.0, and draws 5A. Initially, the robot was equipped with a PandaBoard ES, possessing a 1.2Ghz dual-core CPU with 1GB DDR3 RAM, SD card storage, and drawing 3A. Due to low memory and high read/write latencies, the latter board struggled to bear the load of even a full Ubuntu installation and was replaced with the XU3. For the visual sensor, a stripped down Microsoft Kinect Model

1473 was mounted onto the quadcopter - removing the sensor housing reduced weight from a heavy four pounds to a very nimble 110g. A Pixhawk PX4 flight controller was also added to the system for actuation control - this allowed the focus of the project to remain on the mapping the component of the robot. Lastly, a MaxSonar sonar sensor was used as an altitude sensor in conjunction with the Pixhawk.

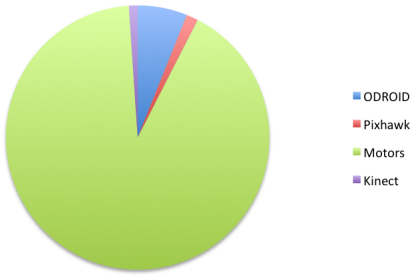
Figure 1: Photograph of finished quadcopter robot.

2.2.2 *Weight Requirements*

As seen in Table 2b, the total weight of the quadcopter robot was approximately 1209g. Each DJI 2212 motor can provide about 600g of thrust, meaning the quadcopter can lift about 2400g maximum. The weight of all components brought the weight to about half that limit, confirming the feasibility of the quadcopter's payload for the tests required.

2.2.3 *Power Requirements*

In order to select a battery, the power consumption of all the components mounted on the robot was calculated. As seen in Figure 2a, the order of components from greatest to least consumption were the DJI 2212 motors, ODROID XU3, Pixhawk, and Kinect sensor. The actuators (and speed controllers) average about 15A during normal operation and the ODROID consumes approximately 4A at full load - since this is engaged in heavy image processing during flight, the ODROID is assumed to be at max. load at all times. The Kinect and Pixhawk also require 700mA and 950mA respectively. In total, the robot consumes about 65.65A during testing operation. In the end, a Zippy 2450mAh 4S LiPo battery was selected to power the robot. To accommodate the 12V, 700mAh requirements of the Kinect as well as the 5V, 5A needs of the ODROID, two power regulators were mounted and installed to the robot's power system, as seen in Table 1.



(a) Power consumption.

Part	Quantity	Weight (g)	Total (g)
Frame	1	282	282
Landing Gear	4	15	60
Motor	4	56	224
ESC	4	30	120
Pixhawk	1	38	38
Battery	1	295	295
Kinect	1	110	110
ODROID XU3	1	80	80
		Total UAV Weight	1209

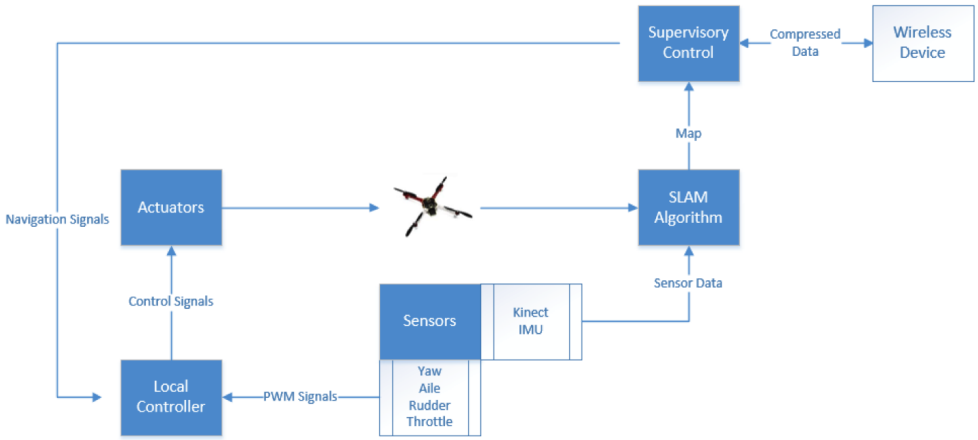
(b) Weights of components

2.3 SOFTWARE

2.3.1 *Overview*

There were a number of key components needed to develop a software architecture for the robot. ROS Hydro was selected as a software package to manage the communication, networking, and overall data representation used throughout the robot, RGBDSLAM_V2 as an algorithm integrated with ROS to develop the 3-D map onboard the robot, and lastly, MAVROS was an additional ROS package that was used to communicate with the Pixhawk controller. The data flow between all the software components can be seen in Figure 5. The cyberphysical design of the robot can be seen below in Figure 3.

Figure 3: Cyber-physical design of robot



2.3.2 *RGBDSLAM_V2*

In order to compute a 3-D map, the quadcopter's onboard computer uses a type of algorithm called SLAM (simultaneous localization and mapping), which computes a map and keeps track of the robotic agent's pose relative to said map. Due to time constraints, it was decided that a proprietary SLAM algorithm was not an option and a previously developed software package was chosen instead. Due to its compatibility with ROS and extensive usage throughout the robotics community, RGBDSLAM_V2 [1] was the software used to produce the map and pose transformations of the robot during flight. As opposed to sensors like a scanning laser, RGBDSLAM_V2 provides a SLAM algorithm that can be interfaced with RGB-D sensors like the Microsoft Kinect used throughout this report. The backend of the algorithm consists of a pose graph, as seen in Figure 4, where each node represents the 3-D pose of the sensor / robot and each edge corresponds to pairwise transformations between sensor poses in the pose graph. This graph is used in tandem with the depth and image data to produce the transformed point clouds in the output map.

2.3.3 *ROS Hydro*

Robot Operating System (ROS) is a meta operating system that is installed as package on a UNIX-based OS like Ubuntu or OS X. In a nutshell, ROS allows one to design their robot using a distributed model, where each ROS *node* is essentially a process or computational component of the robot. It also provides asynchronous and synchronous communication libraries, networking libraries, visualization tools, and supports multiple programming languages. Using said libraries, *nodes* can obtain information output from other nodes via a subscription or reply-request model as well as output data themselves in the same fashion. Usage of ROS in the project allowed focus to remain on algorithm development and abstracts out the software "glue" that connects all the different algorithms running onboard. Within the system presented in the report, there were several nodes that partitioned the processes running in real-time on the robot. RGBDSLAM produced the map and pose of the robot onboard, which were each broadcast to the server when running cooperative mapping. The Map Fusion node was also running on the server (discussed in a later section) which subscribed to said data and produced the global map for all the robots.

2.3.4 *MAVROS*

In order for the onboard computer to communicate with the Pixhawk controller, it makes use of a ROS package called MAVROS. This runs as another ROS node in the ROS system, which allows the computer to send commands to the Pixhawk from a high-level. Such commands include arming and disarming of the motors, take-off and landing commands, checking battery levels, and sending RC values to change the state of the motors (move). MAVROS communicates with the computer using a data protocol called MAVLINK - this protocol allows a graphical user interface, such as the APMPlanner software recommended by the Pixhawk manufacturer, to interface with MAVROS and debug or visualize the internal data that it publishes (IMU data, etc). As explained in the next section, a main function of MAVROS in the project was to command the quadcopter to maintain its altitude during experiments, remotely arm or disarm the robot, etc. This can also been noted in Figure 5.

Figure 4: Data flow of RGBDSLAM [1]

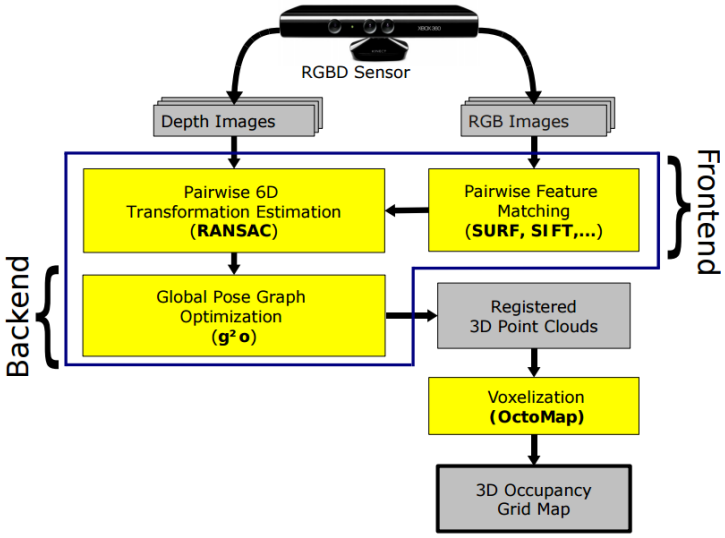
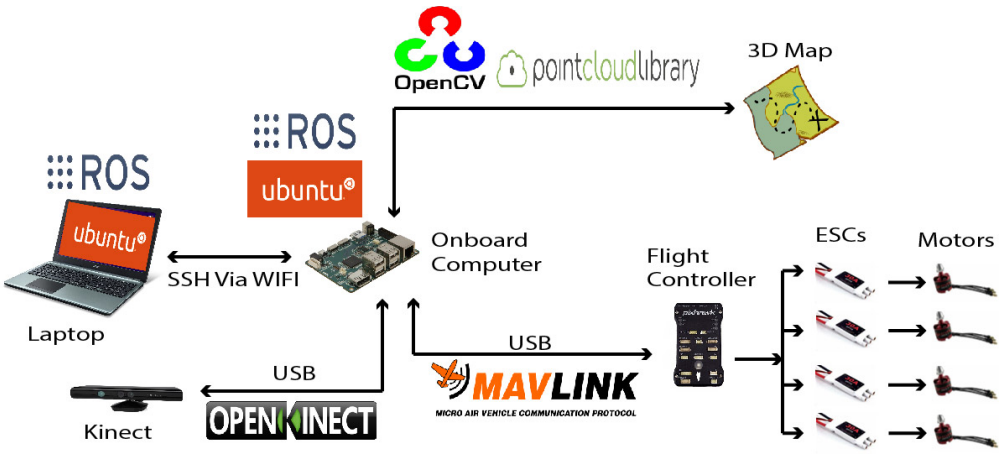


Figure 5: Data flow of quadcopter components.



3

SINGLE ROBOT

3.1 OVERVIEW

For a single robot, this project focused on complete onboard mapping without relying on communication between an external sensor or computer. To demonstrate this, experiments were conducted that involved saving the map for post-flight retrieval and providing an optional stream of data in real-time to a local computer. As the focus of this project is on the mapping component, fully-developed autonomous navigation is not present. However, some elements to facilitate stable flight are implemented and provide a baseline for future work in navigation.

3.2 STABILITY

The Pixhawk PX4 controller provides a built-in PID controller, which is used for drift stability and basic feedback control of the quadcopter. The proportional, integral, and derivative parameters were tuned through trial and error, adjusting each component until the quadcopter's movement no longer oscillated and remained stable. The Pixhawk PX4 also provides an altitude holding system, given an appropriate range sensor and input holding altitude. A sonar sensor was attached to the robot and aimed at the floor to provide an approximate distance to the ground - this data was run through a simple filter and then input to the Pixhawk controller. In tandem, these two elements provided both reasonable drift corrections and the ability for the robot to remain at a set altitude in stable conditions. As a result, conducting flights was much easier and such stability also provides a good foundation for navigational components.

3.3 MAPPING

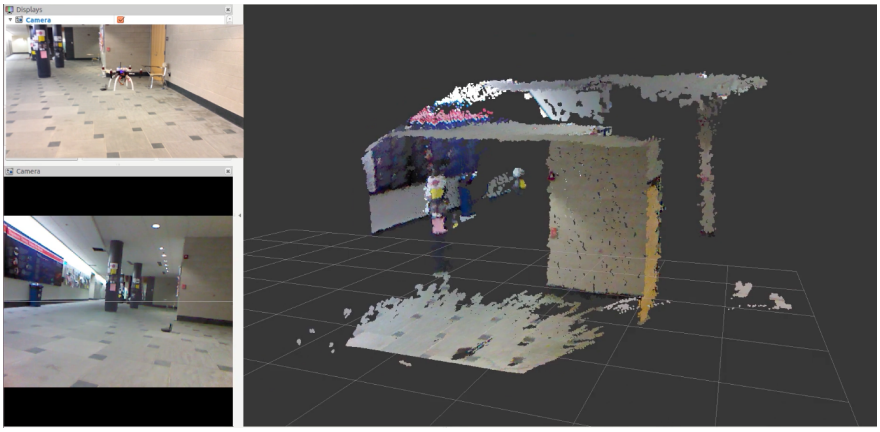
In a previous paper, Bachrach et al. implemented RGB-D mapping, and in a separate publication [2], laser mapping, by offloading real-time processing to a local server [10]. This robot design differs from its contemporaries by computing all steps of the mapping procedure onboard. As explained previously,

the quadcopter's onboard computer runs an RGBDSLAM ROS node in real-time during flight. This node generates the three-dimensional map, which can be be saved on disk as a file through a command in ROS and retrieved after a test flight. The ROS node also allows requests for the current map to be sent over the network to another computer. As a result of difficulties discussed in the result section, a custom package called *Online Mapping* was written that allows for real-time visualization (this is explained in a later section) - this allows the map be streamed efficiently to a nearby computer via Wi-Fi connection. These two options allow for absolute flexibility in situations where remote access to the robot may or may not be possible.

3.4 RESULTS

Testing of a single quadcopter robot was done in the concourse of the Information Technology Engineering building on the UConn Storrs campus. As mentioned in the overview, there were two primary goals to check feasibility of the onboard mapping process - if the map could be produced, saved, and retrieved after the flight and if the mapping process could be (optionally) streamed to a laptop located nearby in real-time. As seen in Figure 6, an image taken of the quadcopter mid-test can be seen in the top left, the RGB stream from the Kinect sensor on the bottom left, and on the right-hand side, the 3-D map generated in real-time by the onboard computer. In all tests, the quadcopter successfully saved the map as a file and was easily retrieved upon completion of the experiment.

Figure 6: Live test of a single quadcopter.



Providing an optional stream to an external laptop proved more difficult. The nature of the RGBDSLAM package made such an operation impossible, as the algorithm, by default, broadcast all the point clouds generated by the mapping procedure to the server at once. This resulted in the point clouds being bottlenecked by the speed of the network router and the sudden massive influx of data to the laptop

caused many clouds to be ignored - the input queue for RVIZ (a ROS visualization tool) is capped and many clouds would be rejected once this was limit was reached. To remedy this issue, the *Online Mapping* component (explained in the next section) was utilized to send only the most recent cloud to the laptop at each timestep. This resulted in a smooth reproduction of the map in real-time over the network in all tests conducted. These tests show that mapping onboard a quadcopter is absolutely possible in real-time. Additionally, the flexibility of map retrieval and optional streaming to nearby computer ensures the robot is robust to environmental circumstances, for instance, if a wireless network is not available for streaming.

4

MULTIPLE ROBOTS

4.1 OVERVIEW

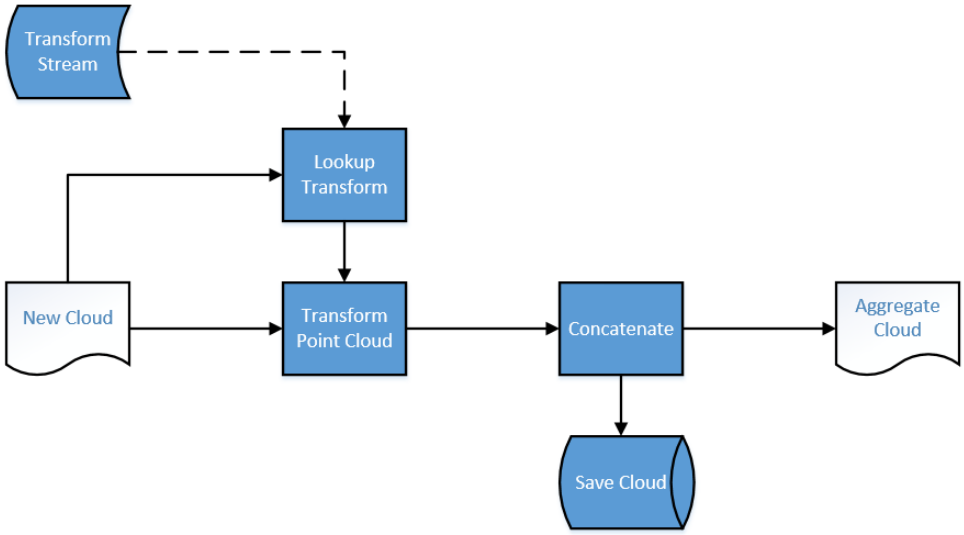
When transitioning from one robotic agent creating its own map to generating a global 3-D map for multiple disjoint robot agents, it's essential to fuse each robot's maps together. This requires the introduction of a centralized, external server as the onboard computational units do not have the computing power to both generate the map for singular robot usage and to construct the cooperative map simultaneously in the case of multiple agents. The server's function is underlined by two key algorithms, first the *Online Mapping* algorithm which reconstructs each robot's map in real-time as it sends timestamped point clouds over the network and second the *Map Fusion* algorithm, which fuses the the two aggregate disjoint maps together.

4.2 ONLINE MAPPING

As discussed in the overview, each quadcopter produces a point cloud segment which it sends over the network to be aggregated into a single map corresponding to that robotic agent. There are two primary methods to accomplish this, both with unique drawbacks. The first method is to deliver the entire aggregated point cloud map computed by RGBDSLAM at given intervals to the server - this becomes infeasible very quickly because as time grows, the size of the map will grow to hundreds of megabytes and current network speeds are unable to process such a load in a reasonable time frame. The other solution is to send each individual point cloud corresponding to a timestep over the network, resulting in a reconstruction of the map on the server. This is inefficient, because the map is then being built twice simultaneously and it is redundant to perform the same operation remotely. However, the experiments in this report used the latter method (despite its obvious weaknesses) given constraints and time restrictions of the project.

Coupled with the mapping component of the RGBDSLAM package, a custom ROS node was written (integrated with RGBDSLAM) that provided a stream of clouds added to the map, each corresponding to new nodes added within the pose graph of RGBDSLAM. As shown in Figure 7, these clouds and their

Figure 7: Data flow of online mapping algorithm.



respective transforms[1] are computed by RGBDSLAM, sent procedurally over the network, received by a server or other external computer, which then executes the online map builder algorithm for each cloud. The algorithm first matches the new cloud $\mathbf{P_n}$ with its transform $\mathbf{T_n}$ by performing a transformation lookup keyed by the timestamp $\mathbf{t_n}$ of the cloud (located in its header data). Once located, the transformation is applied to the cloud, changing its pose to be relative to the aggregate map $\mathbf{P_a}$. Once this transformation is applied, the new cloud can be simply appended to the current aggregate cloud in the simple operation $\mathbf{P_a} += \mathbf{P_n}$ - this new version of the cloud $\mathbf{P_a}$ is then saved into a database or filesystem for the next iteration.

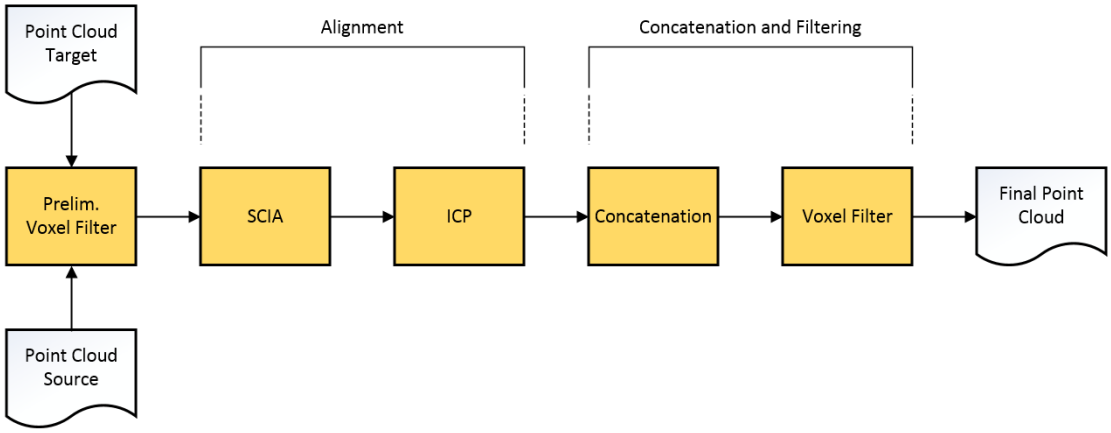
4.3 MAP FUSION

Once these aggregate maps corresponding to the robotic agents are reconstructed on the server, each map has its own point cloud and transformation in 3-D space. Unlike the *Online Mapping* algorithm, there is not a known transformation[2] between the frames of the disjoint maps - this means the *Map Fusion* procedure must approximate the transformation between the two maps. Following a similar approach to Henry et. al. [11], the algorithm transforms two maps into the same global reference frame through several alignment approximations, stitches the maps together, and eliminates redundant data points by applying a voxel

[1] The transform of the cloud is that between the RGB optical frame of the Kinect camera and the fixed frame of the robot's map.
[2] In Online Mapping, the fixed frame is determined by the pose of the first cloud generated, so a transformation can be found from the camera frame of subsequent maps. In Map Fusion, no such initial transformation is known.

filtering algorithm. By incrementally applying the map fusion algorithm to the aggregate maps, the server outputs a timestamped representation of the global map. As shown in Figure 8, this algorithm is divided into two key sections. Unfortunately, this algorithm relies heavily on the assumption that the environment has both lots of unique features to distinguish itself and also that mapped segments by each quadcopter have sufficient overlap.

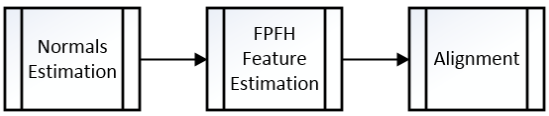
Figure 8: Data flow of map fusion algorithm.



4.3.1 *Sample Consensus Initial Alignment*

As mentioned in the overview, it's necessary to transform the source map into the reference frame of the target map, however, this process must be approximated through an iterative process. The point clouds are first put through an initial alignment phase that provides a rougher, larger-scale transformation estimation. This is done using the Sample Consensus Initial Alignment algorithm [12]. from the Point Cloud Library. This process outputs a final transformation which is applied to the source cloud.

Figure 9: Data flow of SCIA process.



Within the SCIA process itself, there are several subprocesses. First, the normals $\mathbf{N_{source}}$ and $\mathbf{N_{target}}$ are found by PCL's normal estimator. These normal clouds represent the estimated surface normals at each point in the cloud. They are then used in PCL's Fast Point Feature Histogram (FPFH) [12, 13] estimator, to generate approximate feature clouds $\mathbf{F_{source}}$ and $\mathbf{F_{target}}$. These feature clouds store feature descriptors

for each point in the cloud, in this case, FPFH features. Although, the theory behind image features is beyond the scope of this report, the FPFH feature approximates the mean curvature of a point's k-neighbors' geometry with a histogram of values - this type of feature allows computation of a feature cloud in asymptotically linear time, giving a powerful but relatively quick feature estimator. Lastly, these feature clouds from this process are used to estimate an initial transformation **T**, which is applied to the initial source cloud to obtain an initial alignment. This gives the clouds a reasonably close placement relative to each other, and allowing optimal usage of second stage of alignment, ICP, to correct their positioning on a finer scale. The parameters used to tune the SCIA process can be seen in Table 2 within the Appendix.

4.3.2 *Iterative Closest Point*

After the clouds have passed through the initial alignment stage, they are locally optimized using the Iterative Closest Point algorithm. ICP takes point clouds $\mathbf{P_{source}}$ and $\mathbf{P_{target}}$, where it then finds the nearest k-neighbor points in $\mathbf{P_{target}}$. By minimizing the sum of squares error, a rigid transformation **T** between $\mathbf{P_{source}}$ and $\mathbf{P_{target}}$ is output. This transformation is applied to $\mathbf{P_{source}}$ which eliminates narrower errors between the two clouds. The parameters used to tune the ICP process can be seen in Table 2 within the Appendix.

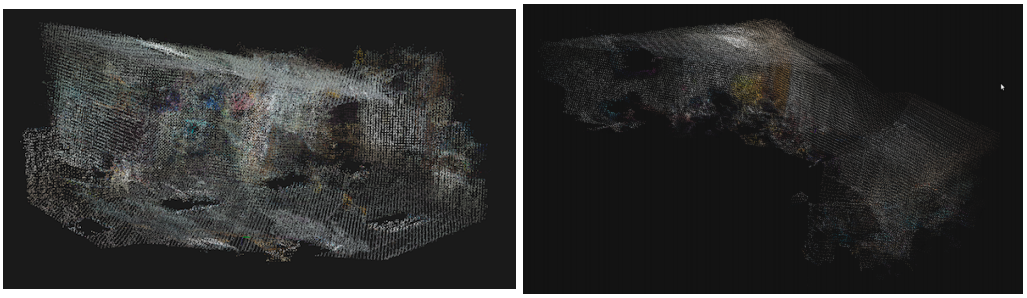
4.3.3 *Filtering and Concatenation*

Lastly, the target and transformed source cloud are concatenated using Point Cloud Library's concatenation features - programmatically, this is as simple as just adding the clouds together. Using another Voxel filter, the excess data points are trimmed away to generate the final stitched version of the map, as shown in Figure 8. The parameters used to tune the filtering process can be seen in Table 2 within the Appendix.

4.4 RESULTS

Due to financial and logistical difficulties, a test involving two quadcopters flying simultaneously in real-time was unable to be conducted within the time frame for this project. As an alternative to demonstrate effectiveness of the approaches demonstrated in this report, two separate flights using the same quadcopter were conducted in the same overlapping environment and saved into *rosbags*. These *rosbags* are a part of

the ROS Hydro software system and allow data to be recorded in real-time and then played back using the same timesteps. The *rosbags* were then simulated as live agents, with map data streaming at the same frequency and conditions as a real multi-robot flight test - this provided data to the map fusion server that almost exactly duplicates two quadcopters flying simultaneously and served as a substitution for a real test.



(a) Bad approximation.

(b) Good approximation.

Figure 10: Visual process of map fusion.

The results presented in Figure 10 were obtained in the L.I.N.K.S Lab by flying the robot around the perimeter of the room. In one test, approximately one half of the room was captured by the first robot and the second half of the room was captured by a second robot, with little to no overlap. This resulted in a very poor approximation to a global map, as seen in Figure 10a. In the second test, the same half of the room was captured by the two robots - this resulted in a very good transformation and approximation to a global map - this is shown in Figure 10b. These results show that even with the powerful sample consensus and iterative closest point algorithms, a significant amount of overlap or matching features is needed in able to successfully compute a global map corresponding to the individual quadcopter robots. Although no live test with two robots was conducted, the success of the simulation provides a good sense of reliability for the *Online Mapping* and *Map Fusion* algorithms - a full live test would simply reinforce the success and failures of these experiments.

5

CONCLUSIONS

5.1 OVERVIEW

This report aimed to show that onboard mapping in a GPS-denied environment using a UAV robot is possible (without any external sensors or communication with another computer) and that multiple robots can successfully contribute to a full 3-D global map that is computed by a nearby centralized server. The experiments shown throughout this report successfully demonstrate these ideas in real indoor locations where external positioning is not available. Robots can successfully explore an environment, for instance, within a communication blackout zone by saving a map to their filesystem that is retrieved post-flight. When a wireless network is conveniently available, they can also show a user what is being explored by streaming their map in real-time to another computer. In the case of multiple robots, results revealed that map building is not limited to a single robot, and with a server or computer nearby, multiple robots can divy up an environment by fusing their explored locations together. Although these concepts are largely limited to indoor environments and where there is enough overlap to distinguish segments of an environment, these tests provide a framework for expansion into a more robust cooperative system or one with a full navigation component.

5.2 FUTURE WORK

As stated earlier, this project did not focus on navigation or control of multiple quadcopters, but rather concentrated on the feasibility of mapping using multiple quadcopters. A natural progression forward would be to implement localization and possibly object avoidance algorithms in parallel with the mapping procedures. The RGBDSLAM algorithm in this report was used mostly for its mapping and pose graph features, but was selected partly because of its capabilities in estimating trajectory and facilitating full autonomous navigation. Implementation of navigation in tandem with the concepts presented in this report would be a significant achievement. In regard to the cooperative mapping component, it's clear that there are limitations to the approach taken in this report. If two maps provided by robotic agents do not

overlap, a global map would not be produced. A potential solution to this problem is to store each map in a database and attempt to randomly fuse them together at intervals, similar to the approach taken by Riazuelo et al [7]. On a higher level, a centralized approach to cooperative mapping would ideally not be necessary at all. Instead, the onboard computational units would cooperatively compute the global map onboard, independent of external communication. Unfortunately, it is clear that the application of such a decentralized approach is limited largely by technological capabilities. A possible direction to overcome to this hurdle could be to add a secondary computational unit aboard each robotic agent - this would allow for sufficient computational overhead to compute the global map in a decentralized manner, but would increase the power and weight requirements of the robot significantly, resulting in need of a larger capacity battery to compensate for increased thrust and power needs.

6

APPENDIX

Table 2: Map Fusion Parameters

Parameter	Value
Voxel Leaf Size	0.05
ICP Max. Corresp. Dist.	0.1
ICP Max. Iter.	50
ICP Trans. Epsilon	1e-8
ICP Euclid. Epsilon	1
SCIA Min. Sample. Dist	0.01
SCIA Max. Corresp. Dist	0.1
SCIA Max. Iter.	50
Radius Normals	0.05
Radius Features	0.10

BIBLIOGRAPHY

[1] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, "An evaluation of the rgb-d slam system," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 1691–1696, IEEE, 2012.

[2] A. Bachrach, S. Prentice, R. He, and N. Roy, "Range–robust autonomous navigation in gps-denied environments," *Journal of Field Robotics*, vol. 28, no. 5, pp. 644–666, 2011.

[3] S. Shen, N. Michael, and V. Kumar, "Autonomous multi-floor indoor navigation with a computationally constrained mav," in *Robotics and automation (ICRA), 2011 IEEE international conference on*, pp. 20–25, IEEE, 2011.

[4] S. P. Soundararaj, A. K. Sujeeth, and A. Saxena, "Autonomous indoor helicopter flight using a single onboard camera," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pp. 5307–5314, IEEE, 2009.

[5] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Visual odometry and mapping for autonomous flight using an rgb-d camera," in *International Symposium on Robotics Research (ISRR)*, pp. 1–16, 2011.

[6] M. Walter and J. Leonard, "An experimental investigation of cooperative slam," in *Proceedings of the Fifth IFAC/EURON Symposium on Intelligent Autonomous Vehicles,(Lisbon, Portugal)*, 2004.

[7] L. Riazuelo, J. Civera, and J. Montiel, "C 2 tam: A cloud framework for cooperative tracking and mapping," *Robotics and Autonomous Systems*, vol. 62, no. 4, pp. 401–413, 2014.

[8] K. Y. Leung, T. D. Barfoot, and H. H. Liu, "Decentralized cooperative slam for sparsely-communicating robot networks: a centralized-equivalent approach," *Journal of Intelligent & Robotic Systems*, vol. 66, no. 3, pp. 321–342, 2012.

[9] A. Cunningham and F. Dellaert, "Large-scale experimental design for decentralized slam," in *SPIE Defense, Security, and Sensing*, pp. 83870O–83870O, International Society for Optics and Photonics, 2012.

[10] A. Bachrach, S. Prentice, R. He, P. Henry, A. S. Huang, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Estimation, planning, and mapping for autonomous flight using an rgb-d camera in gps-denied environments," *The International Journal of Robotics Research*, vol. 31, no. 11, pp. 1320–1343, 2012.

[11] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 647–663, 2012.

[12] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (fpfh) for 3d registration," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pp. 3212–3217, IEEE, 2009.

[13] R. B. Rusu, A. Holzbach, N. Blodow, and M. Beetz, "Fast geometric point labeling using conditional random fields," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pp. 7–12, IEEE, 2009.